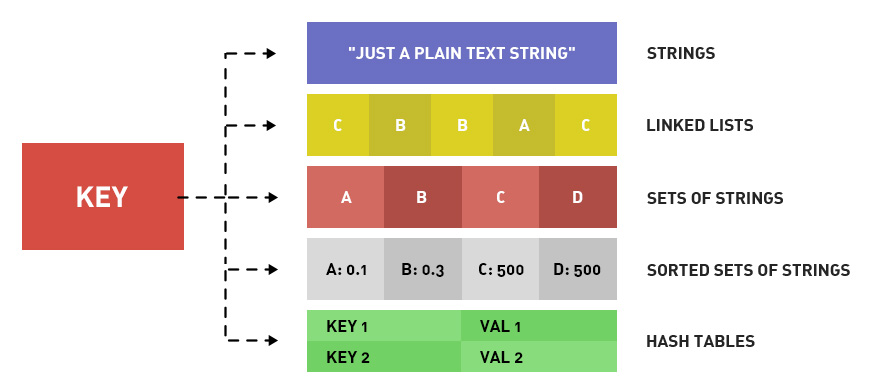
# Redis数据类型



## string 字符串

字符串类型是 Redis 最基础的数据结构，首先键是字符串类型，而且其他几种结构都是在字符串类型基础上构建的。字符串类型实际上可以是字符串：简单的字符串、XML、JSON；数字：整数、浮点数；二进制：图片、音频、视频。

## Hash

在 Redis中哈希类型是指键本身是一种键值对结构，如 value={{field1,value1},……{fieldN,valueN}} 使用场景：哈希结构相对于字符串序列化缓存信息更加直观，并且在更新操作上更加便捷。所以常常用于用户信息等管理，但是哈希类型和关系型数据库有所不同，哈希类型是稀疏的，而关系型数据库是完全结构化的，关系型数据库可以做复杂的关系查询，而 Redis 去模拟关系型复杂查询开发困难且维护成本高。

### Hash冲突操作

Redis 通过链式哈希解决冲突：也就是同一个 桶里面的元素使用链表保存。但是当链表过长就会导致查找性能变差可能，所以 Redis 为了追求快，使用了两个全局哈希表。用于 rehash 操作，增加现有的哈希桶数量，减少哈希冲突。

开始默认使用 「hash 表 1 」保存键值对数据，「hash 表 2」 此刻没有分配空间。当数据越来越多触发 rehash 操作，则执行以下操作： 给 「hash 表 2 」分配更大的空间； 将 「hash 表 1 」的数据重新映射拷贝到 「hash 表 2」 中； 释放 「hash 表 1」 的空间。

值得注意的是，将 hash 表 1 的数据重新映射到 hash 表 2 的过程中并不是一次性的，这样会造成 Redis 阻塞，无法提供服务。 而是采用了渐进式 rehash，每次处理客户端请求的时候，先从「 hash 表 1」 中第一个索引开始，将这个位置的 所有数据拷贝到 「hash 表 2」 中，就这样将 rehash 分散到多次请求过程中，避免耗时阻塞。

## List（列表）

列表类型是用来储存多个有序的字符串，列表中的每个字符串成为元素，一个列表最多可以储存 2 ^ 32 – 1 个元素，在 Redis 中，可以队列表两端插入和弹出，还可以获取指定范围的元素列表、获取指定索引下的元素等，列表是一种比较灵活的数据结构，它可以充当栈和队列的角色。

## Set（集合）

集合类型也是用来保存多个字符串的元素，但和列表不同的是集合中不允许有重复的元素，并且集合中的元素是无序的，不能通过索引下标获取元素，Redis 除了支持集合内的增删改查，同时还支持多个集合取交集、并集、差集。合理的使用好集合类型，能在实际开发中解决很多实际问题。

## zset（sorted set：\*\*\*\*有序集合）

有序集合和集合有着必然的联系，它保留了集合不能有重复成员的特性，但不同得是，有序集合中的元素是可以排序的，但是它和列表的使用索引下标作为排序依据不同的是：它给每个元素设置一个分数，作为排序的依据

# 内存淘汰策略

## 过期算法

volatile-lru：从已设置过期时间的数据集（server. db[i]. expires）中挑选最近最少使用的数据淘汰；

volatile-ttl：从已设置过期时间的数据集（server. db[i]. expires）中挑选将要过期的数据淘汰。

volatile-random：从已设置过期时间的数据集（server. db[i]. expires）中任意选择数据淘汰。

allkeys-lru：从数据集（server. db[i]. dict）中挑选最近最少使用的数据淘汰。

allkeys-random：从数据集（server. db[i]. dict）中任意选择数据淘汰。

no-enviction（驱逐）：禁止驱逐数据。

## 过期删除策略

过期策略通常有以下三种：

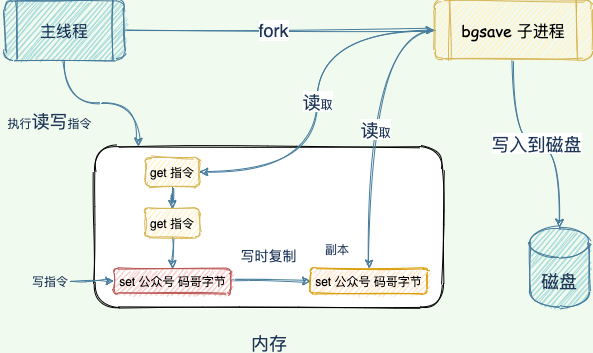
* 定时过期：每个设置过期时间的key都需要创建一个定时器，到过期时间就会立即清除。该策略可以立即清除过期的数据，对内存很友好；但是会占用大量的CPU资源去处理过期的数据，从而影响缓存的响应时间和吞吐量。
* 惰性过期：只有当访问一个key时，才会判断该key是否已过期，过期则清除。该策略可以最大化地节省CPU资源，却对内存非常不友好。极端情况可能出现大量的过期key没有再次被访问，从而不会被清除，占用大量内存。
* 定期过期：每隔一定的时间，会扫描一定数量的数据库的expires字典中一定数量的key，并清除其中已过期的key。该策略是前两者的一个折中方案。通过调整定时扫描的时间间隔和每次扫描的限定耗时，可以在不同情况下使得CPU和内存资源达到最优的平衡效果。 (expires字典会保存所有设置了过期时间的key的过期时间数据，其中，key是指向键空间中的某个键的指针，value是该键的毫秒精度的UNIX时间戳表示的过期时间。键空间是指该Redis集群中保存的所有键。)

# Redis持久化的方式

## RDB

RDB 是 Redis DataBase 的缩写。按照一定的时间周期策略把内存的数据以快照的形式保存到硬盘的二进制文件。即 Snapshot 快照存储，对应产生的数据文件为 dump.rdb，通过配置文件中的 save 参数来定义快照的周期。核心函数：rdbSave（生成 RDB 文件）和 rdbLoad（从文件加载内存）两个函数。

Redis 使用操作系统的多进程写时复制技术 COW(Copy On Write) 来实现快照持久化，保证数据一致性。 Redis 在持久化时会调用 glibc 的函数fork产生一个子进程，快照持久化完全交给子进程来处理，父进程继续处理客户端请求。 当主线程执行写指令修改数据的时候，这个数据就会复制一份副本， bgsave 子进程读取这个副本数据写到 RDB 文件。 这既保证了快照的完整性，也允许主线程同时对数据进行修改，避免了对正常业务的影响。



## AOF

AOF 是 Append-only file 的缩写。Redis会将每一个收到的写命令都通过 Write 函数追加到文件最后，类似于 MySQL 的 binlog。当 Redis 重启是会通过重新执行文件中保存的写命令来在内存中重建整个数据库的内容。每当执行服务器（定时）任务或者函数时，flushAppendOnlyFile 函数都会被调用， 这个函数执行以下两个工作： WRITE：根据条件，将 aof\_buf 中的缓存写入到 AOF 文件； SAVE：根据条件，调用 fsync 或 fdatasync 函数，将 AOF 文件保存到磁盘中。

## 区别

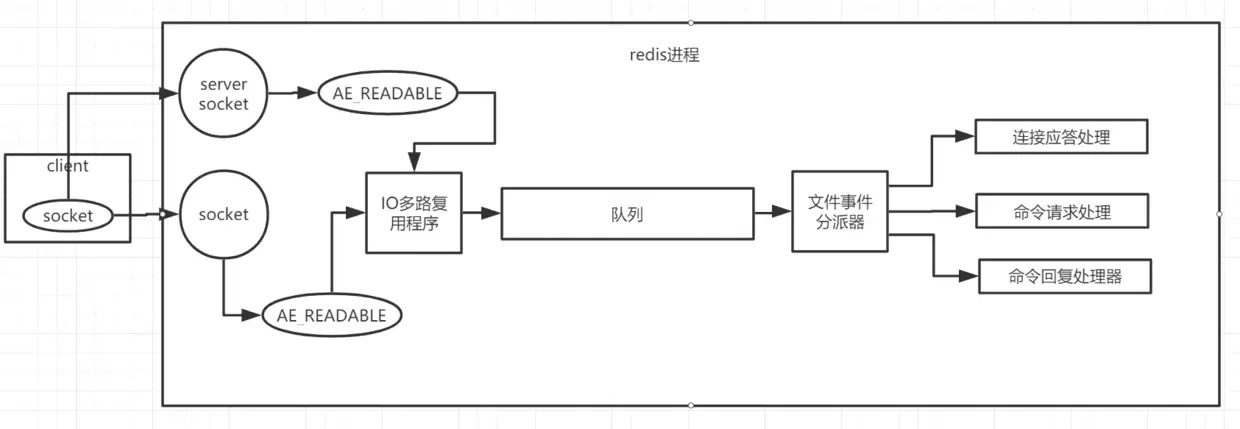
RDB 和 AOF 的区别：

* AOF 文件比 RDB 更新频率高，优先使用 AOF 还原数据；
* AOF比 RDB 更安全也更大；
* RDB 性能比 AOF 好； 如果两个都配了优先加载 AOF。

# IO模型

## 6.0之前

Redis 是基于 reactor 模式开发了网络事件处理器，这个处理器叫做文件事件处理器（file event handler）。由于这个文件事件处理器是单线程的，所以 Redis 才叫做单线程的模型。采用 IO 多路复用机制同时监听多个 Socket，根据 socket 上的事件来选择对应的事件处理器来处理这个事件。



文件事件处理器的结构包含了四个部分：

1、多个 Socket。Socket 会产生 AE\_READABLE 和 AE\_WRITABLE 事件： 当 socket 变得可读时或者有新的可以应答的 socket 出现时，socket 就会产生一个 AE\_READABLE 事件 当 socket 变得可写时，socket 就会产生一个 AE\_WRITABLE 事件。

2、IO 多路复用程序

3、文件事件分派器

4、事件处理器。事件处理器包括：连接应答处理器、命令请求处理器、命令回复处理器，每个处理器对应不同的 socket 事件：

* 如果是客户端要连接 Redis，那么会为 socket 关联连接应答处理器
* 如果是客户端要写数据到 Redis（读、写请求命令），那么会为 socket 关联命令请求处理器
* 如果是客户端要从 Redis 读数据，那么会为 socket 关联命令回复处理器

多个 socket 会产生不同的事件，不同的事件对应着不同的操作，IO 多路复用程序监听着这些 Socket，当这些 Socket 产生了事件，IO 多路复用程序会将这些事件放到一个队列中，通过这个队列，以有序、同步、每次一个事件的方式向文件时间分派器中传送。当事件处理器处理完一个事件后，IO 多路复用程序才会继续向文件分派器传送下一个事件。